DIEPSTRATEN 16-2-2
Context Controller Having Instruction-Based Time Slice Task Switching Capability and Processor Employing the Same
Attorney: J. Joel Justiss (972-480-8800)   Serial No. 09/213,970
Replacement Sheet

FIG. 1

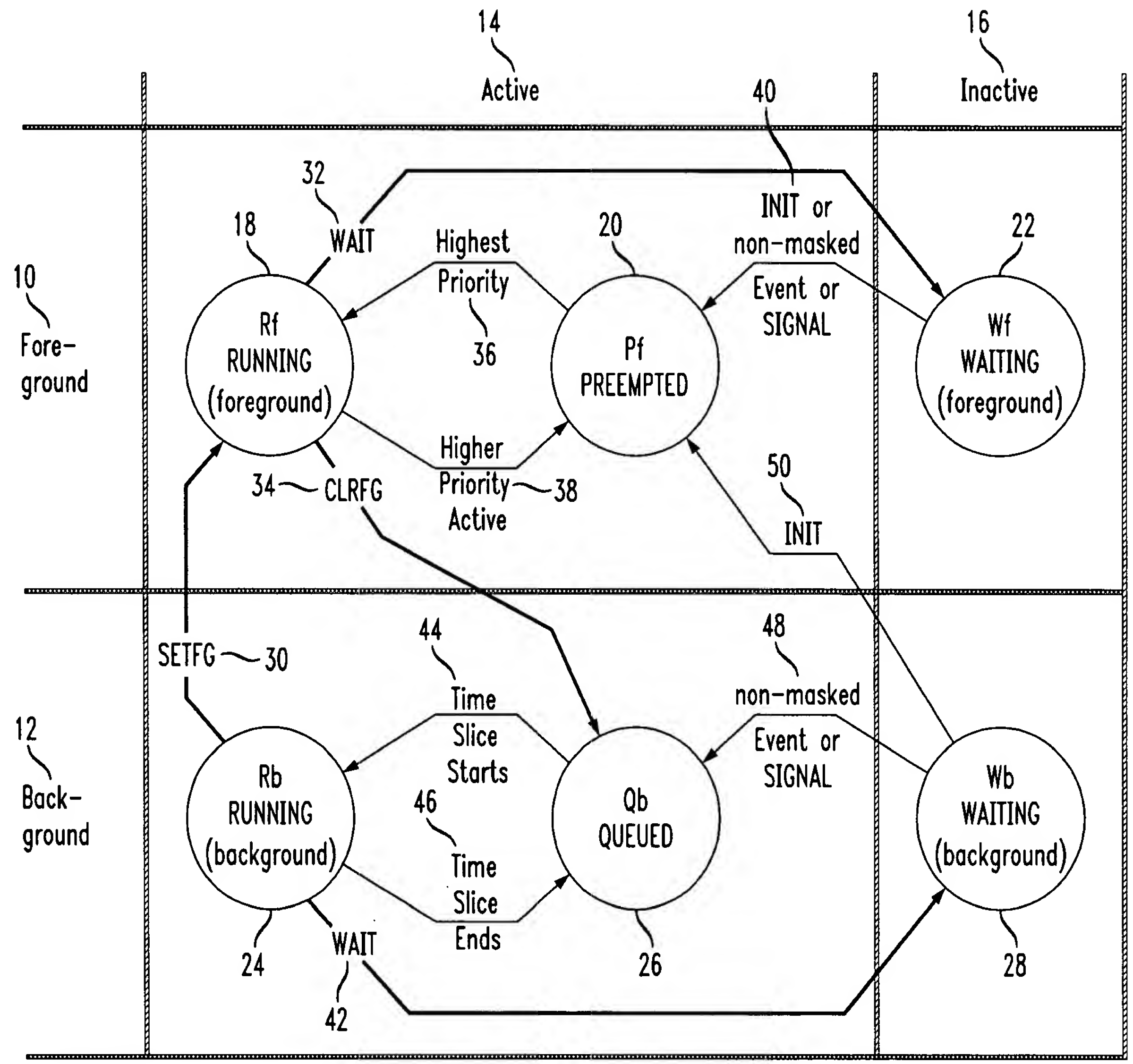


FIG. 2

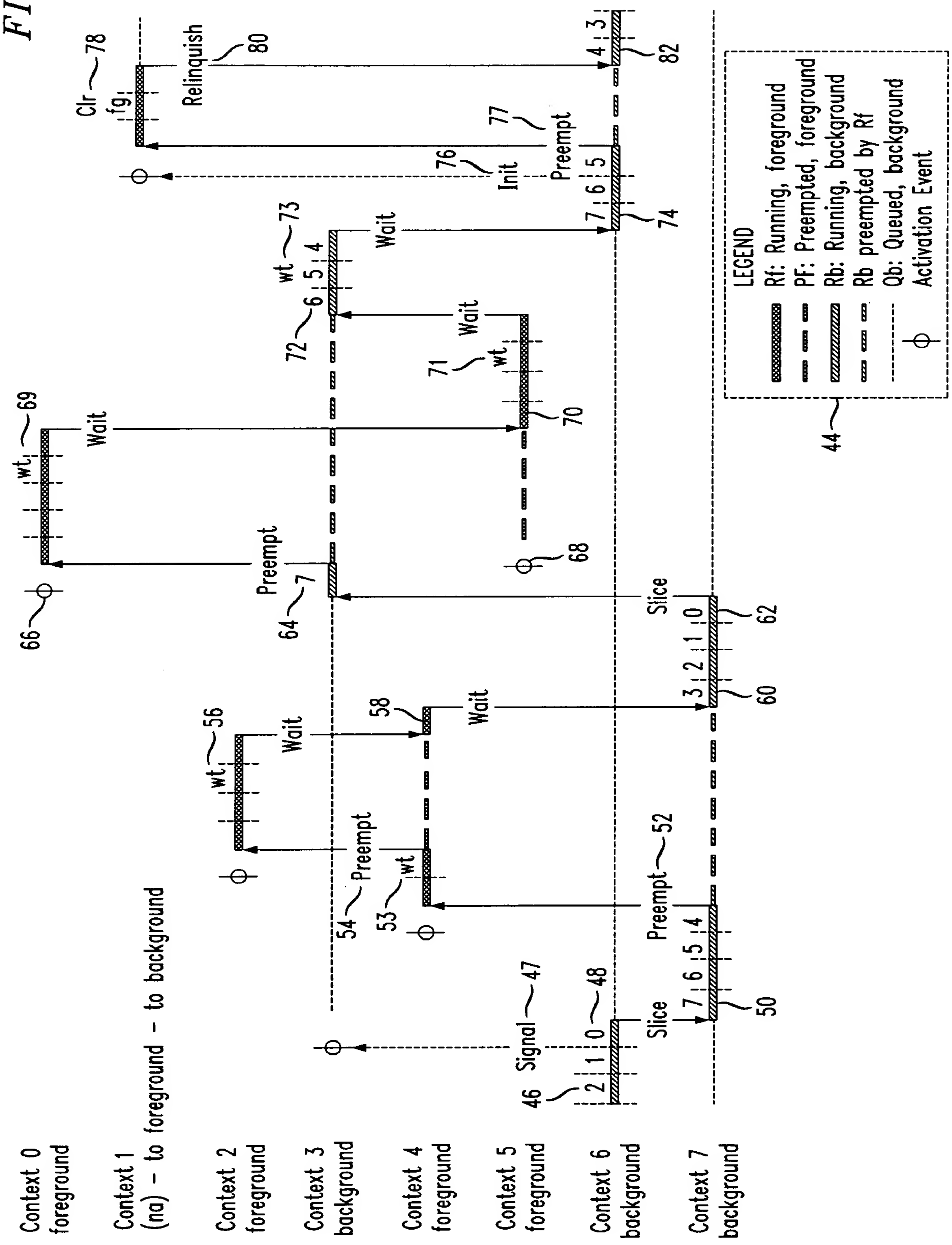

Context 0 foreground
Context 1 (na) - to foreground - to background
Context 2 foreground
Context 3 background
Context 4 foreground
Context 5 foreground
Context 6 background
Context 7 background
46
2 1 0
Signal
47
48
Slice
7 6 5 4
50
Preempt
52
53
wt
54
Preempt
56
wt
Wait
58
Wait
3 2 1 0
60
62
Slice
64
7
Preempt
66
68
69
wt
Wait
70
71
wt
Wait
72
6 5 4
wt
73
Wait
7 6 5
74
76
Init
Preempt
77
Clr
78
fg
Relinquish
80
4 3
82
LEGEND
Rf: Running, foreground
PF: Preempted, foreground
Rb: Running, background
Rb preempted by Rf
Qb: Queued, background
Activation Event
44

FIG. 3

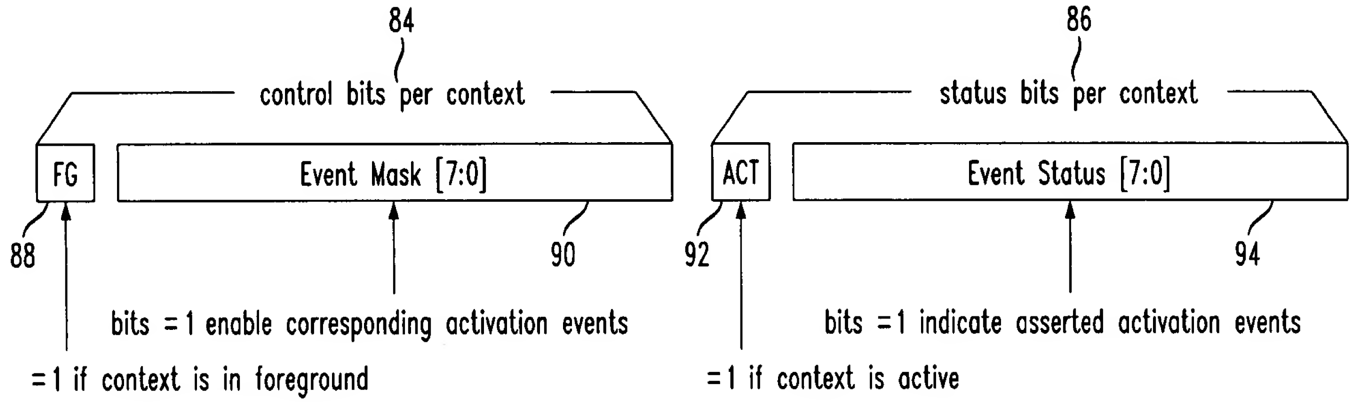
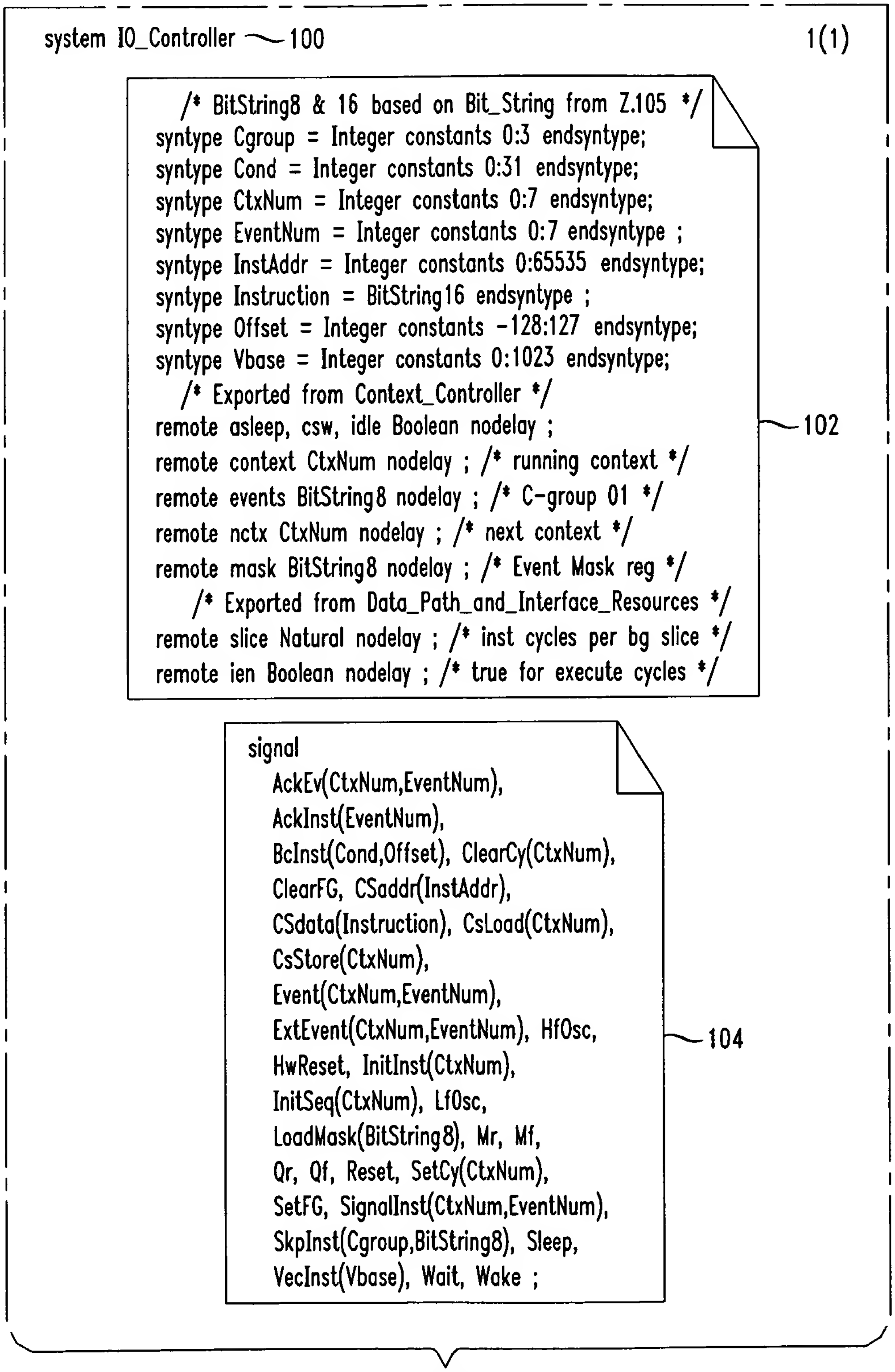

84
control bits per context
FG
Event Mask [7:0]
88
90
bits = 1 enable corresponding activation events
= 1 if context is in foreground
86
status bits per context
ACT
Event Status [7:0]
92
94
bits = 1 indicate asserted activation events
= 1 if context is active

FIG. 4A



```
  /* BitString8 & 16 based on Bit_String from Z.105 */
syntype Cgroup = Integer constants 0:3 endsyntype;
syntype Cond = Integer constants 0:31 endsyntype;
syntype CtxNum = Integer constants 0:7 endsyntype;
syntype EventNum = Integer constants 0:7 endsyntype ;
syntype InstAddr = Integer constants 0:65535 endsyntype;
syntype Instruction = BitString16 endsyntype ;
syntype Offset = Integer constants -128:127 endsyntype;
syntype Vbase = Integer constants 0:1023 endsyntype;
  /* Exported from Context_Controller */
remote asleep, csw, idle Boolean nodelay ;
remote context CtxNum nodelay ; /* running context */
remote events BitString8 nodelay ; /* C-group 01 */
remote nctx CtxNum nodelay ; /* next context */
remote mask BitString8 nodelay ; /* Event Mask reg */
   /* Exported from Data_Path_and_Interface_Resources */
remote slice Natural nodelay ; /* inst cycles per bg slice */
remote ien Boolean nodelay ; /* true for execute cycles */
```

```
signal
   AckEv(CtxNum,EventNum),
   AckInst(EventNum),
   BcInst(Cond,Offset), ClearCy(CtxNum),
   ClearFG, CSaddr(InstAddr),
   CSdata(Instruction), CsLoad(CtxNum),
   CsStore(CtxNum),
   Event(CtxNum,EventNum),
   ExtEvent(CtxNum,EventNum), HfOsc,
   HwReset, InitInst(CtxNum),
   InitSeq(CtxNum), LfOsc,
   LoadMask(BitString8), Mr, Mf,
   Qr, Qf, Reset, SetCy(CtxNum),
   SetFG, SignalInst(CtxNum,EventNum),
   SkpInst(Cgroup,BitString8), Sleep,
   VecInst(Vbase), Wait, Wake ;
```

TO FIG. 4B

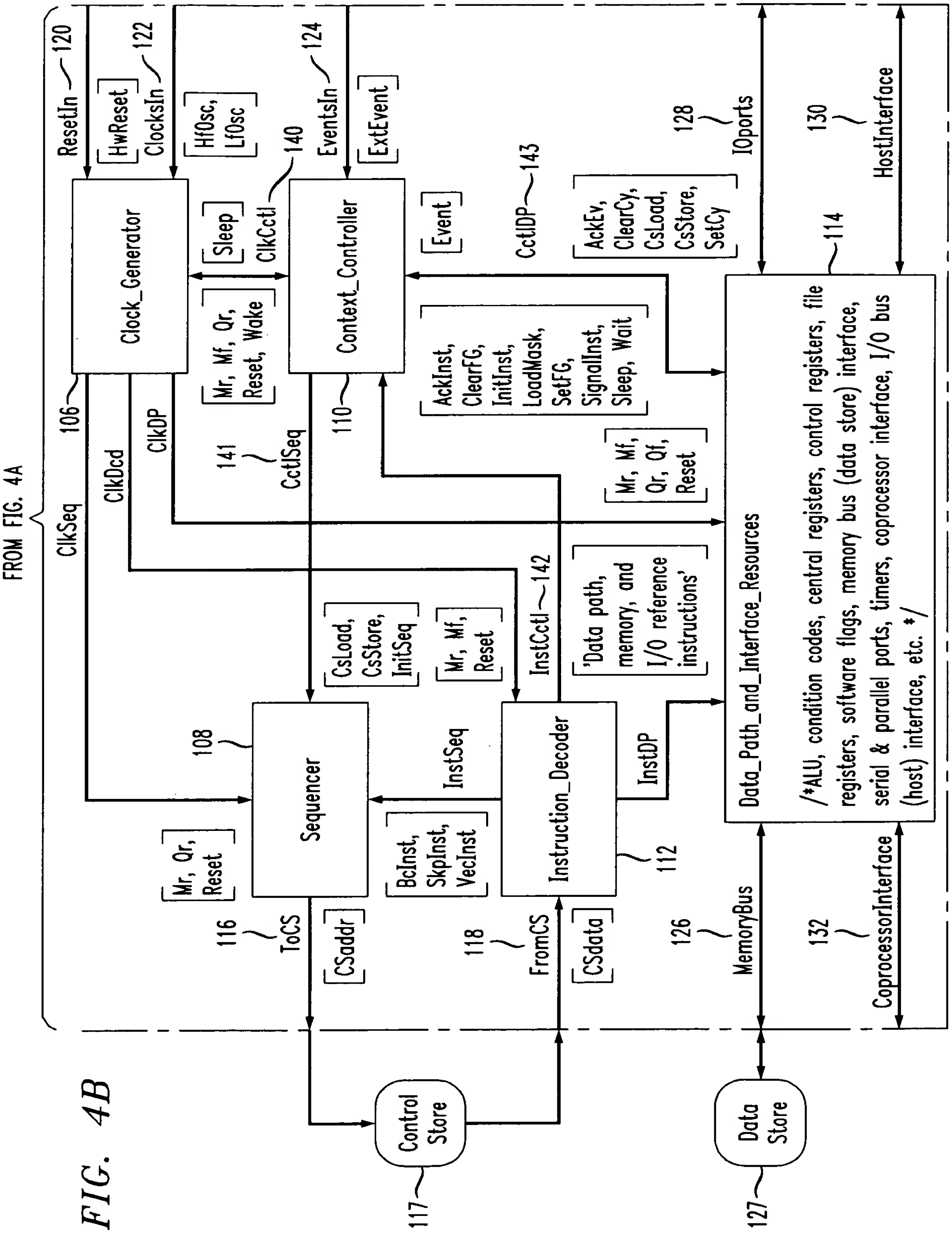

FROM FIG. 4A
ResetIn
120
HwReset
ClocksIn
122
HfOsc,
LfOsc
EventsIn
124
ExtEvent
Clock_Generator
106
Sleep
ClkCctl
140
Mr, Mf, Qr,
Reset, Wake
Context_Controller
110
Event
CctlDP
143
AckEv,
ClearCy,
CsLoad,
CsStore,
SetCy
128
IOports
130
HostInterface
AckInst,
ClearFG,
InitInst,
LoadMask,
SetFG,
SignalInst,
Sleep, Wait
ClkSeq
ClkDcd
ClkDP
141
CctlSeq
Mr, Mf,
Qr, Qf,
Reset
114
'Data path,
memory, and
I/O reference
instructions'
InstCctl
142
CsLoad,
CsStore,
InitSeq
Mr, Mf,
Reset
108
Sequencer
InstSeq
Instruction_Decoder
InstDP
Data_Path_and_Interface_Resources
/*ALU, condition codes, central registers, control registers, file
registers, software flags, memory bus (data store) interface,
serial & parallel ports, timers, coprocessor interface, I/O bus
(host) interface, etc. */
Mr, Qr,
Reset
BcInst,
SkpInst,
VecInst
112
116
ToCS
CSaddr
118
FromCS
CSdata
126
MemoryBus
132
CoprocessorInterface
Control
Store
117
Data
Store
127

FIG. 4B

FIG. 5

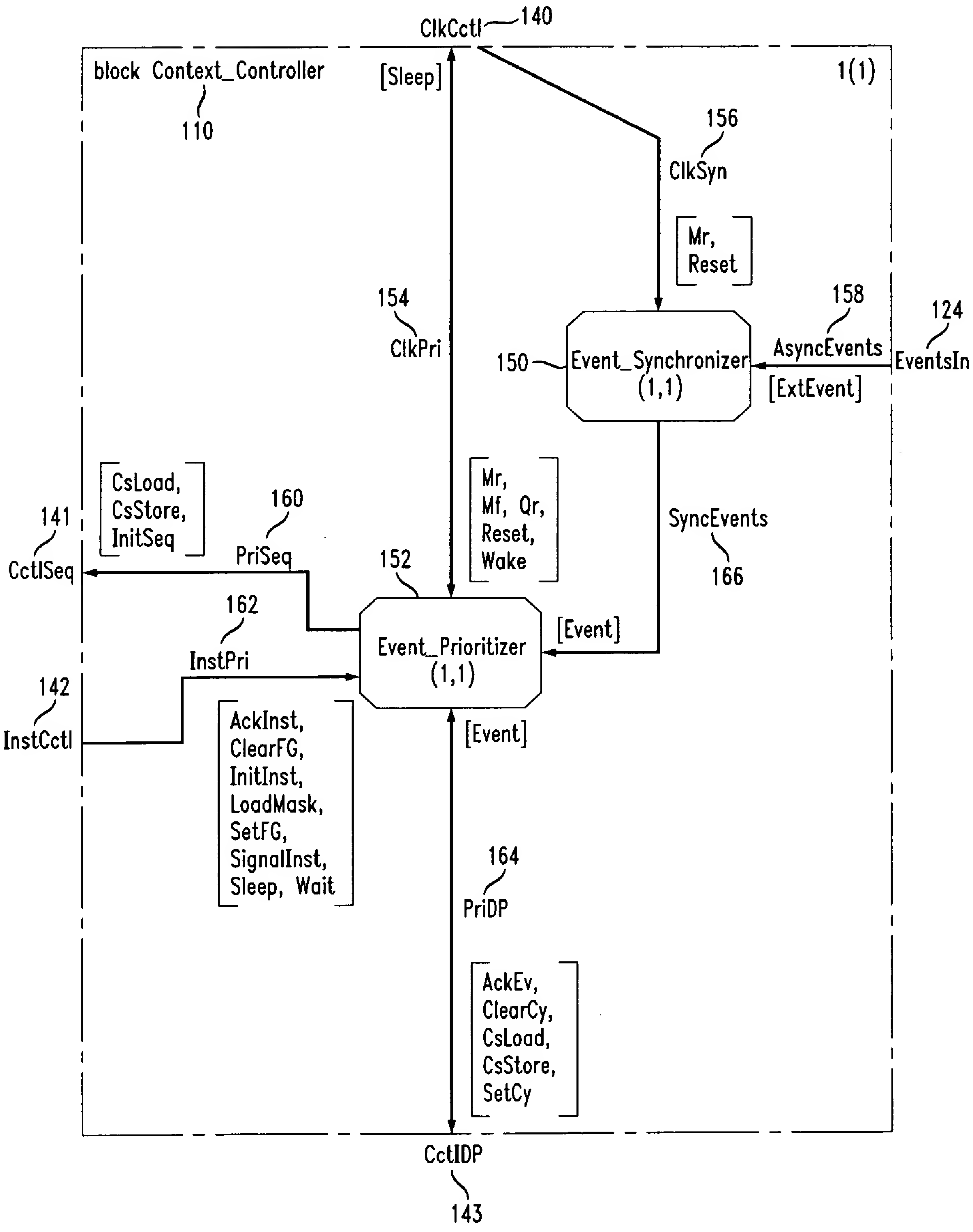
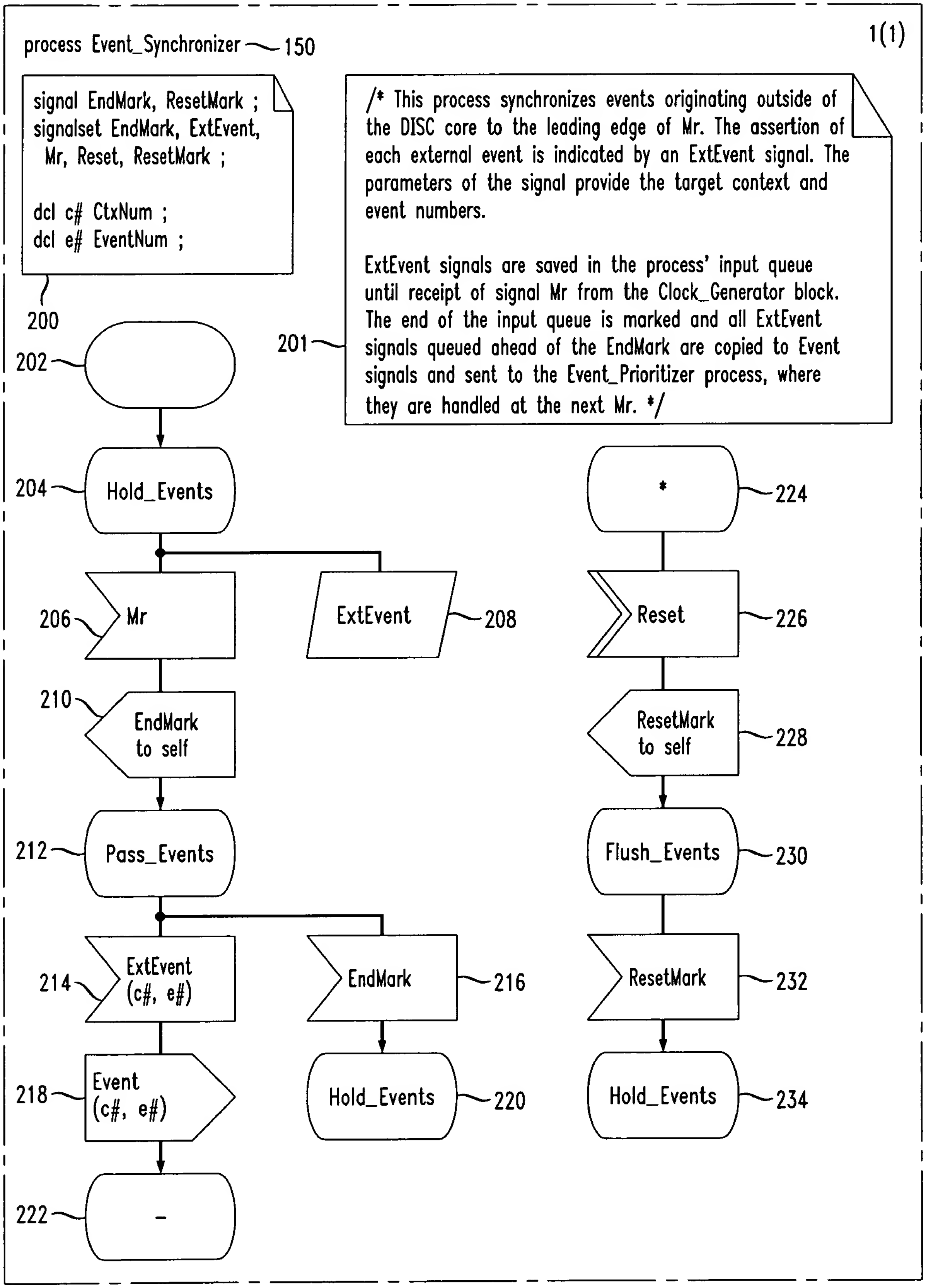

ClkCctl
140
block Context_Controller
110
[Sleep]
1(1)
156
ClkSyn
Mr,
Reset
154
ClkPri
158
AsyncEvents
124
EventsIn
150
Event_Synchronizer
(1,1)
[ExtEvent]
CsLoad,
CsStore,
InitSeq
141
CctlSeq
160
PriSeq
152
Mr,
Mf, Qr,
Reset,
Wake
SyncEvents
166
162
InstPri
[Event]
Event_Prioritizer
(1,1)
142
InstCctl
AckInst,
ClearFG,
InitInst,
LoadMask,
SetFG,
SignalInst,
Sleep, Wait
[Event]
164
PriDP
AckEv,
ClearCy,
CsLoad,
CsStore,
SetCy
CctlDP
143

FIG. 6

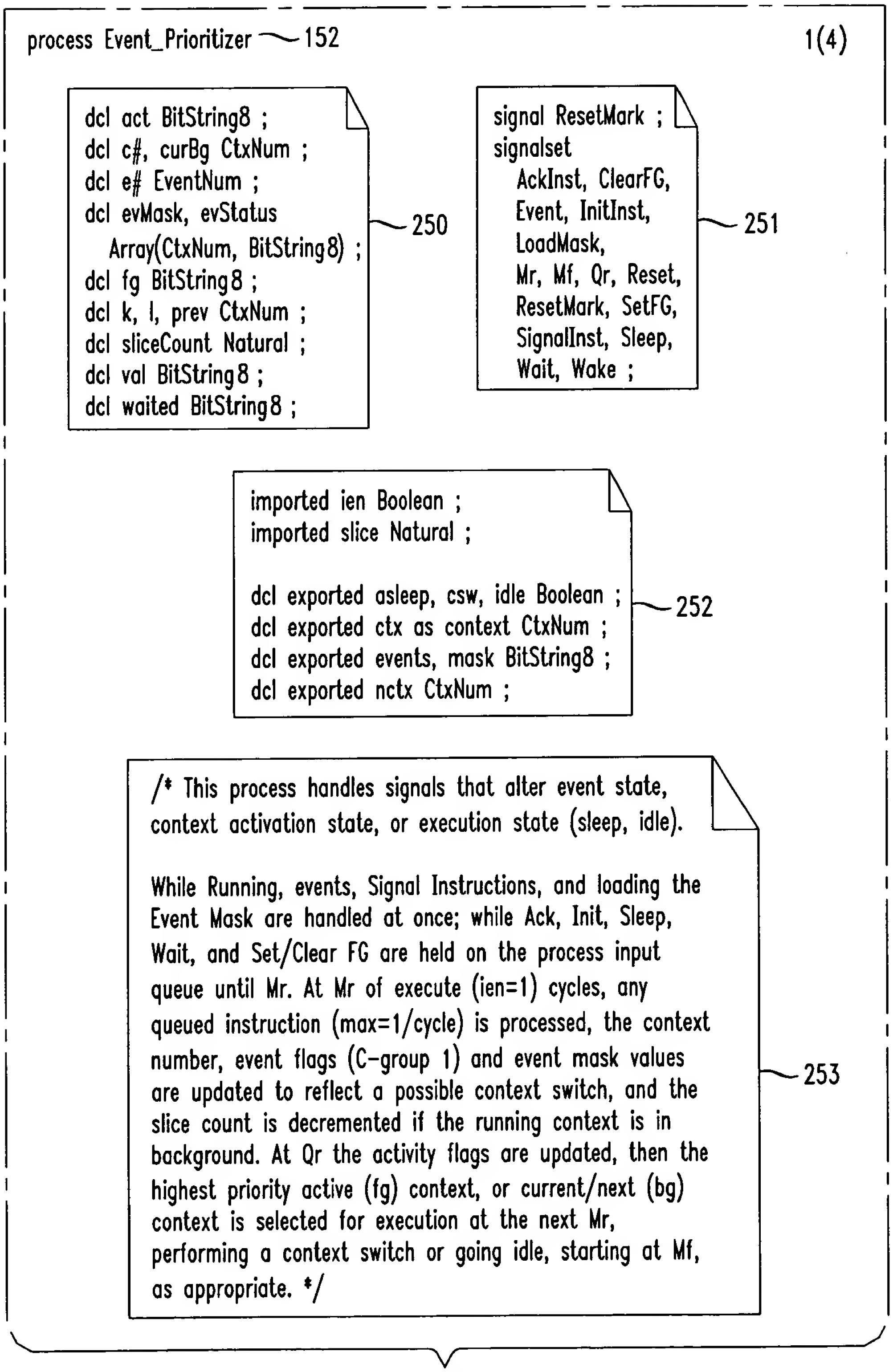


```
signal EndMark, ResetMark ;
signalset EndMark, ExtEvent,
 Mr, Reset, ResetMark ;

dcl c# CtxNum ;
dcl e# EventNum ;
```

/* This process synchronizes events originating outside of the DISC core to the leading edge of Mr. The assertion of each external event is indicated by an ExtEvent signal. The parameters of the signal provide the target context and event numbers.

ExtEvent signals are saved in the process' input queue until receipt of signal Mr from the Clock_Generator block. The end of the input queue is marked and all ExtEvent signals queued ahead of the EndMark are copied to Event signals and sent to the Event_Prioritizer process, where they are handled at the next Mr. */

FIG. 7A1



```
dcl act BitString8 ;
dcl c#, curBg CtxNum ;
dcl e# EventNum ;
dcl evMask, evStatus
  Array(CtxNum, BitString8) ;
dcl fg BitString8 ;
dcl k, l, prev CtxNum ;
dcl sliceCount Natural ;
dcl val BitString8 ;
dcl waited BitString8 ;
```

```
signal ResetMark ;
signalset
  AckInst, ClearFG,
  Event, InitInst,
  LoadMask,
  Mr, Mf, Qr, Reset,
  ResetMark, SetFG,
  SignalInst, Sleep,
  Wait, Wake ;
```

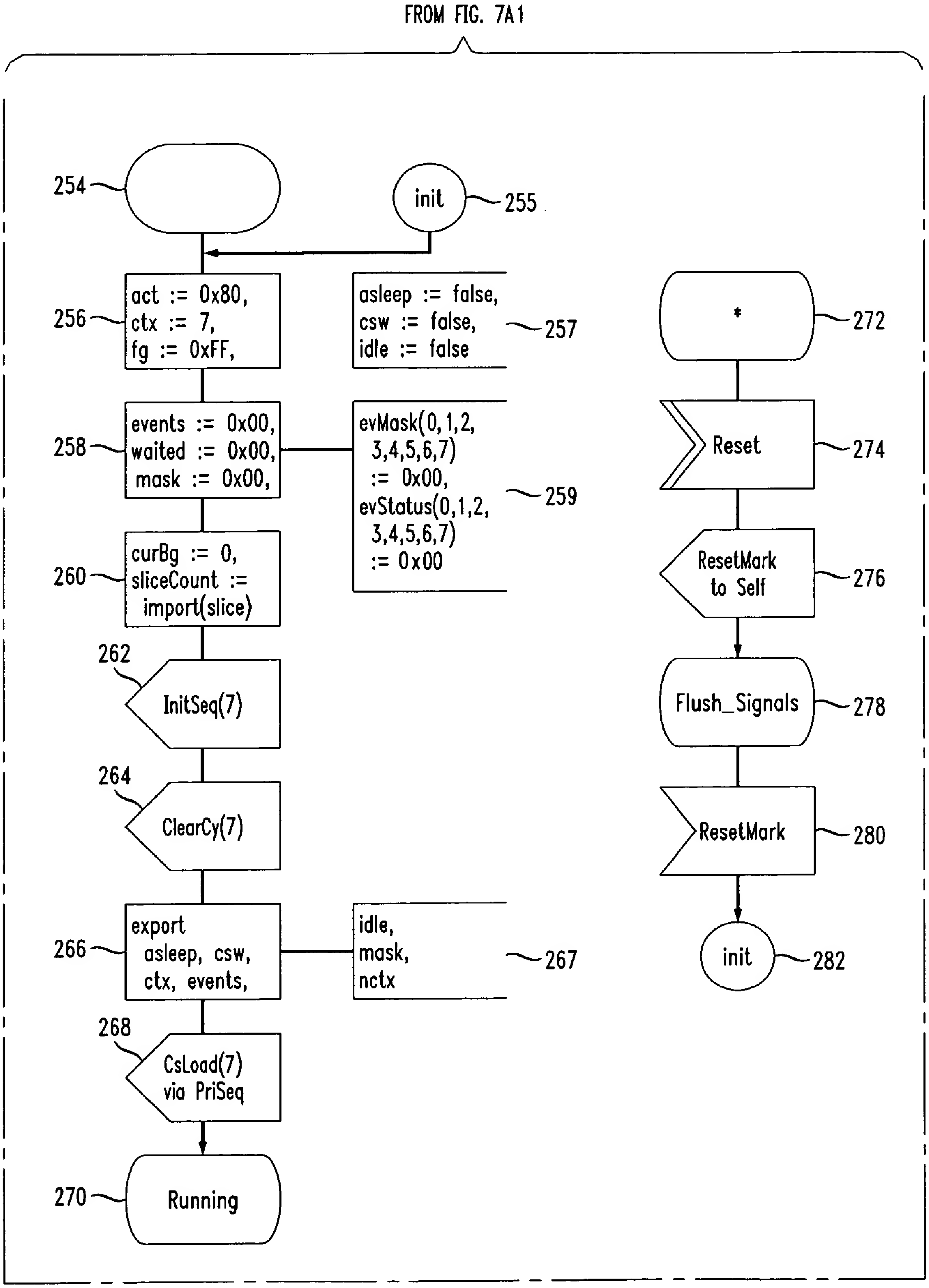
```
imported ien Boolean ;
imported slice Natural ;

dcl exported asleep, csw, idle Boolean ;
dcl exported ctx as context CtxNum ;
dcl exported events, mask BitString8 ;
dcl exported nctx CtxNum ;
```

/* This process handles signals that alter event state, context activation state, or execution state (sleep, idle).

While Running, events, Signal Instructions, and loading the Event Mask are handled at once; while Ack, Init, Sleep, Wait, and Set/Clear FG are held on the process input queue until Mr. At Mr of execute (ien=1) cycles, any queued instruction (max=1/cycle) is processed, the context number, event flags (C-group 1) and event mask values are updated to reflect a possible context switch, and the slice count is decremented if the running context is in background. At Qr the activity flags are updated, then the highest priority active (fg) context, or current/next (bg) context is selected for execution at the next Mr, performing a context switch or going idle, starting at Mf, as appropriate. */

TO FIG. 7A2

FIG. 7A2


FROM FIG. 7A1
254
init
255
256
act := 0x80,
ctx := 7,
fg := 0xFF,
asleep := false,
csw := false,
idle := false
257
258
events := 0x00,
waited := 0x00,
mask := 0x00,
evMask(0,1,2,
3,4,5,6,7)
:= 0x00,
evStatus(0,1,2,
3,4,5,6,7)
:= 0x00
259
260
curBg := 0,
sliceCount :=
import(slice)
262
InitSeq(7)
264
ClearCy(7)
266
export
asleep, csw,
ctx, events,
idle,
mask,
nctx
267
268
CsLoad(7)
via PriSeq
270
Running
*
272
Reset
274
ResetMark
to Self
276
Flush_Signals
278
ResetMark
280
init
282

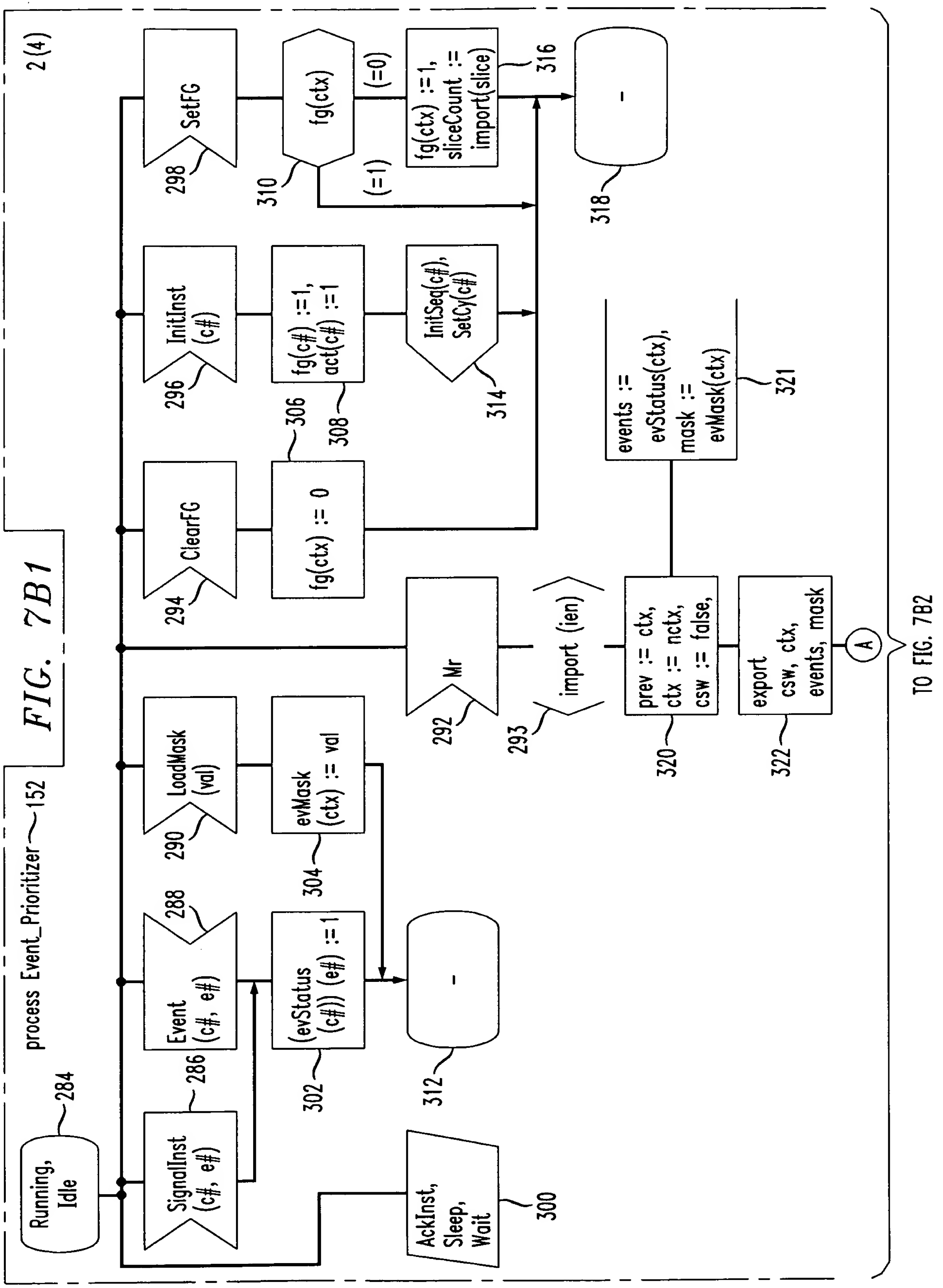

process Event_Prioritizer
152
2(4)
Running, Idle
284
SignalInst (c#, e#)
286
Event (c#, e#)
288
LoadMask (val)
290
Mr
292
ClearFG
294
InitInst (c#)
296
SetFG
298
AckInst, Sleep, Wait
300
(evStatus (c#)) (e#) :=1
302
evMask (ctx) := val
304
fg(ctx) := 0
306
fg(c#) :=1, act(c#) :=1
308
fg(ctx)
310
(=1)
(=0)
–
312
InitSeq(c#), SetCy(c#)
314
fg(ctx) :=1, sliceCount := import(slice)
316
–
318
import (ien)
293
prev := ctx, ctx := nctx, csw := false,
320
events := evStatus(ctx), mask := evMask(ctx)
321
export csw, ctx, events, mask
322
A
TO FIG. 7B2

FIG. 7B1

FIG. 7B2

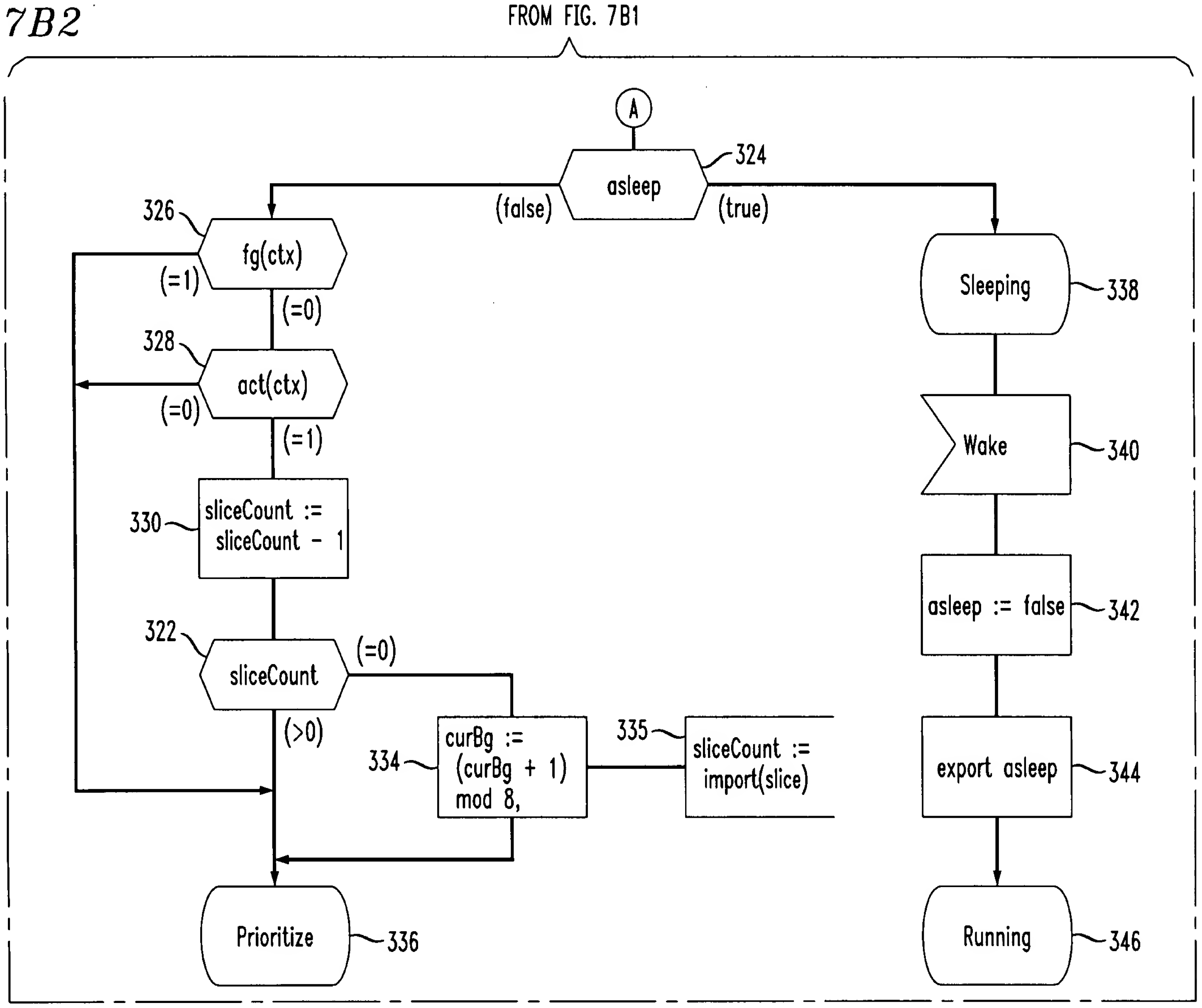

FROM FIG. 7B1
A
asleep
324
(false)
(true)
326
fg(ctx)
(=1)
(=0)
328
act(ctx)
(=0)
(=1)
330
sliceCount :=
sliceCount - 1
322
sliceCount
(=0)
(>0)
334
curBg :=
(curBg + 1)
mod 8,
335
sliceCount :=
import(slice)
Prioritize
336
Sleeping
338
Wake
340
asleep := false
342
export asleep
344
Running
346

FIG. 7C

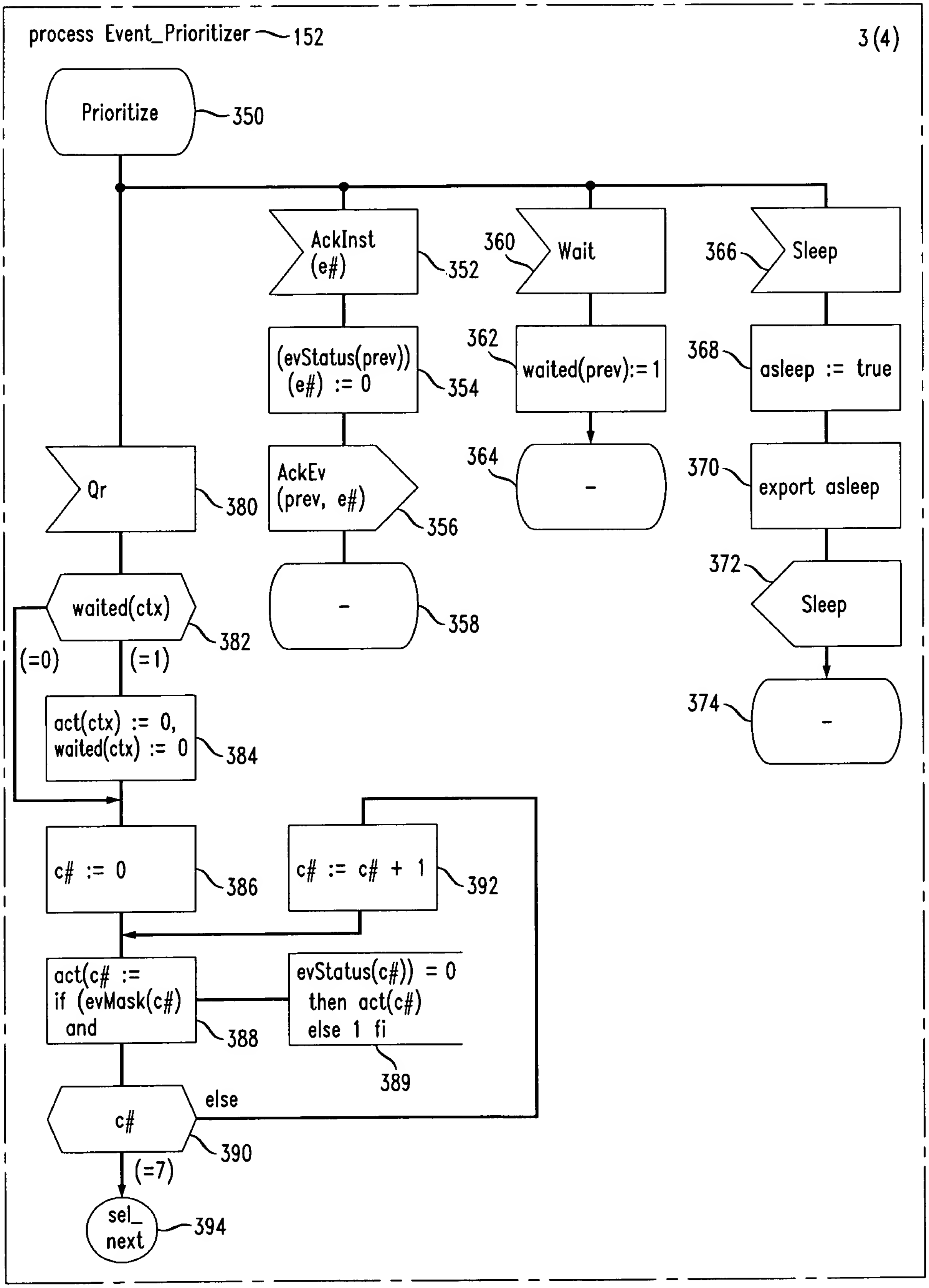
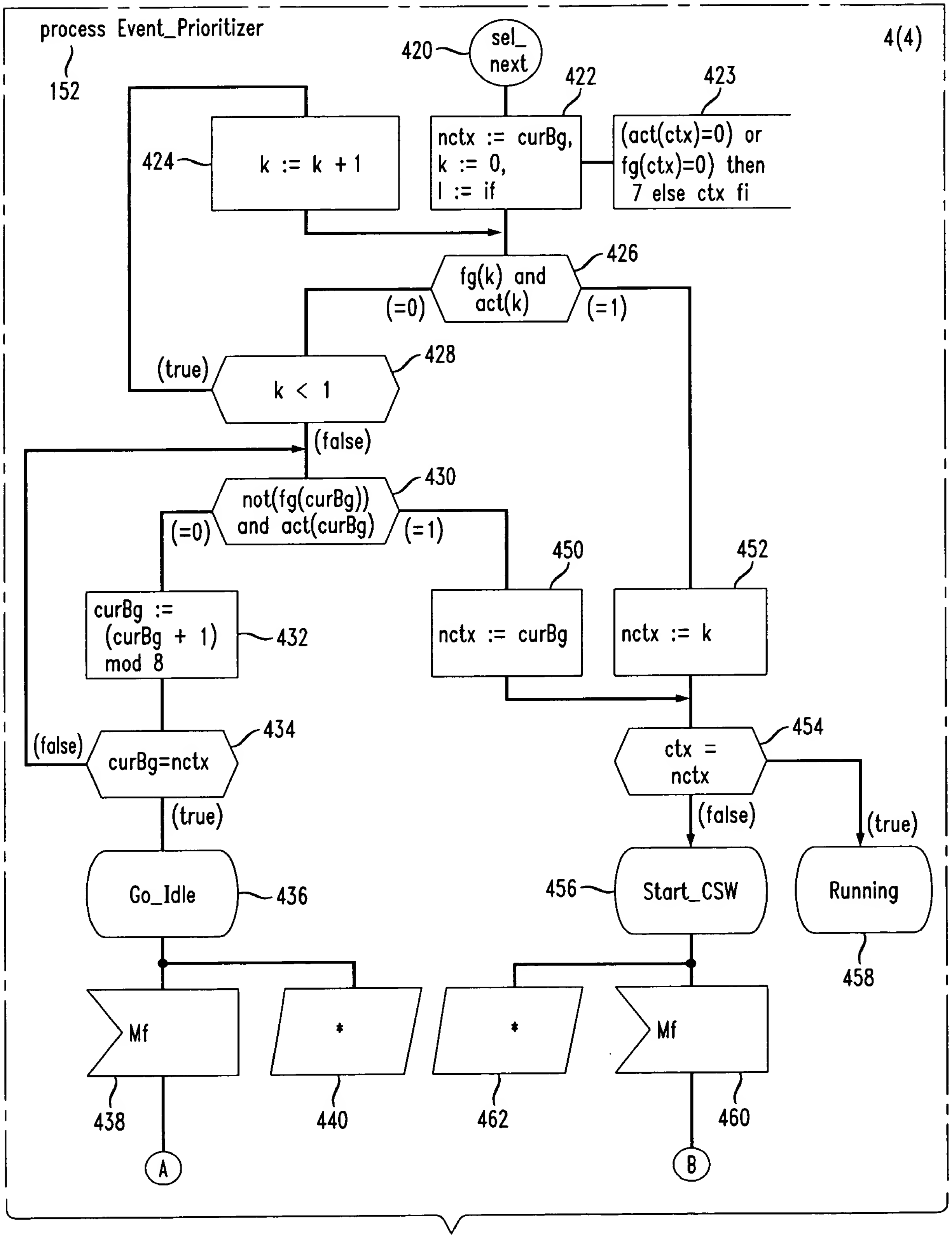

process Event_Prioritizer
152
3(4)
Prioritize
350
AckInst (e#)
352
360
Wait
366
Sleep
(evStatus(prev)) (e#) := 0
354
362
waited(prev):=1
368
asleep := true
Qr
380
AckEv (prev, e#)
356
364
-
370
export asleep
372
Sleep
waited(ctx)
382
-
358
(=0)
(=1)
374
-
act(ctx) := 0, waited(ctx) := 0
384
c# := 0
386
c# := c# + 1
392
act(c# := if (evMask(c#) and
388
evStatus(c#)) = 0 then act(c#) else 1 fi
389
else
c#
390
(=7)
sel_ next
394

FIG. 7D1



TO FIG. 7D2

FIG. 7D2

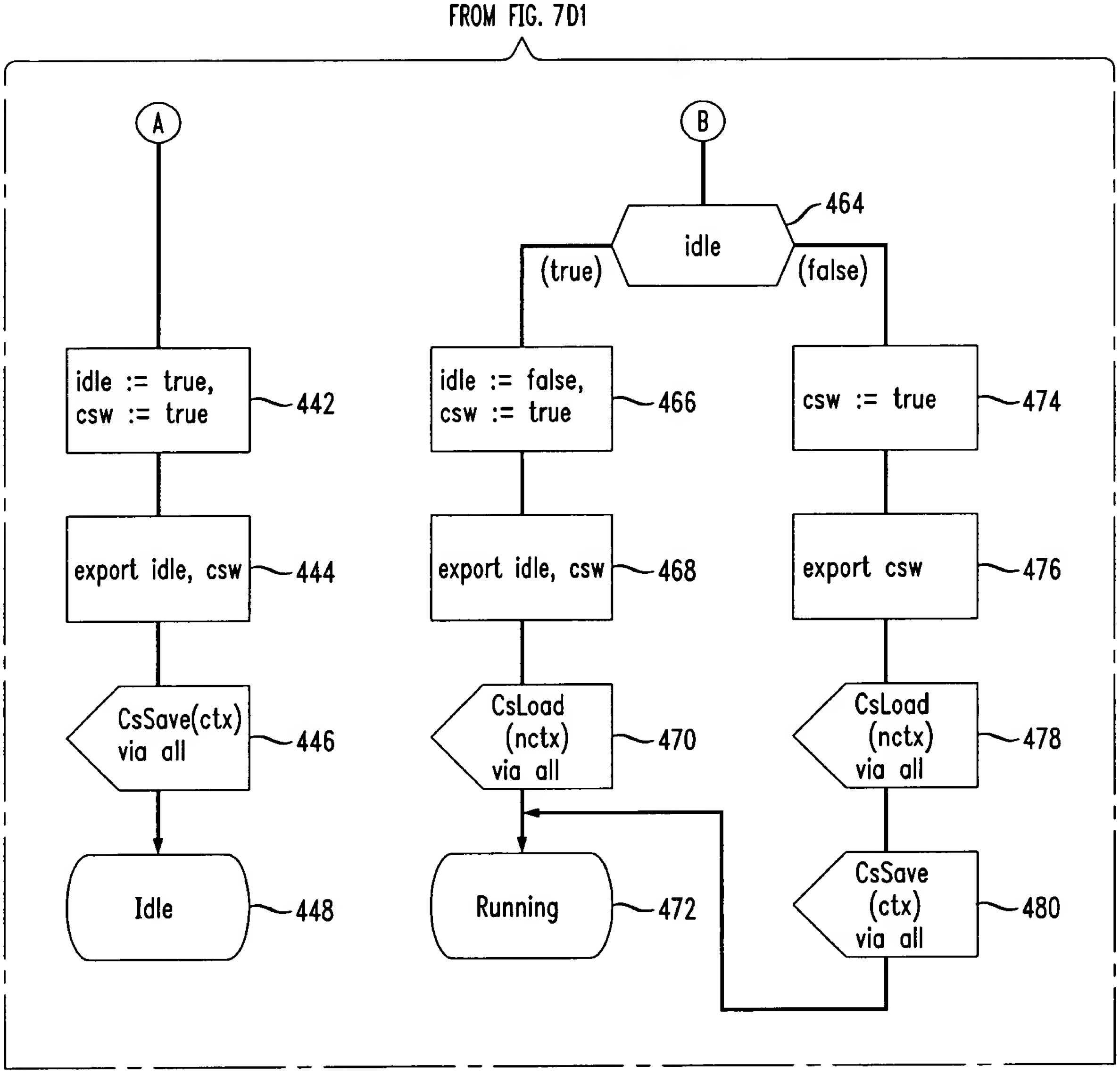

FROM FIG. 7D1
A
B
idle
464
(true)
(false)
idle := true,
csw := true
442
idle := false,
csw := true
466
csw := true
474
export idle, csw
444
export idle, csw
468
export csw
476
CsSave(ctx)
via all
446
CsLoad
(nctx)
via all
470
CsLoad
(nctx)
via all
478
Idle
448
Running
472
CsSave
(ctx)
via all
480

FIG. 8

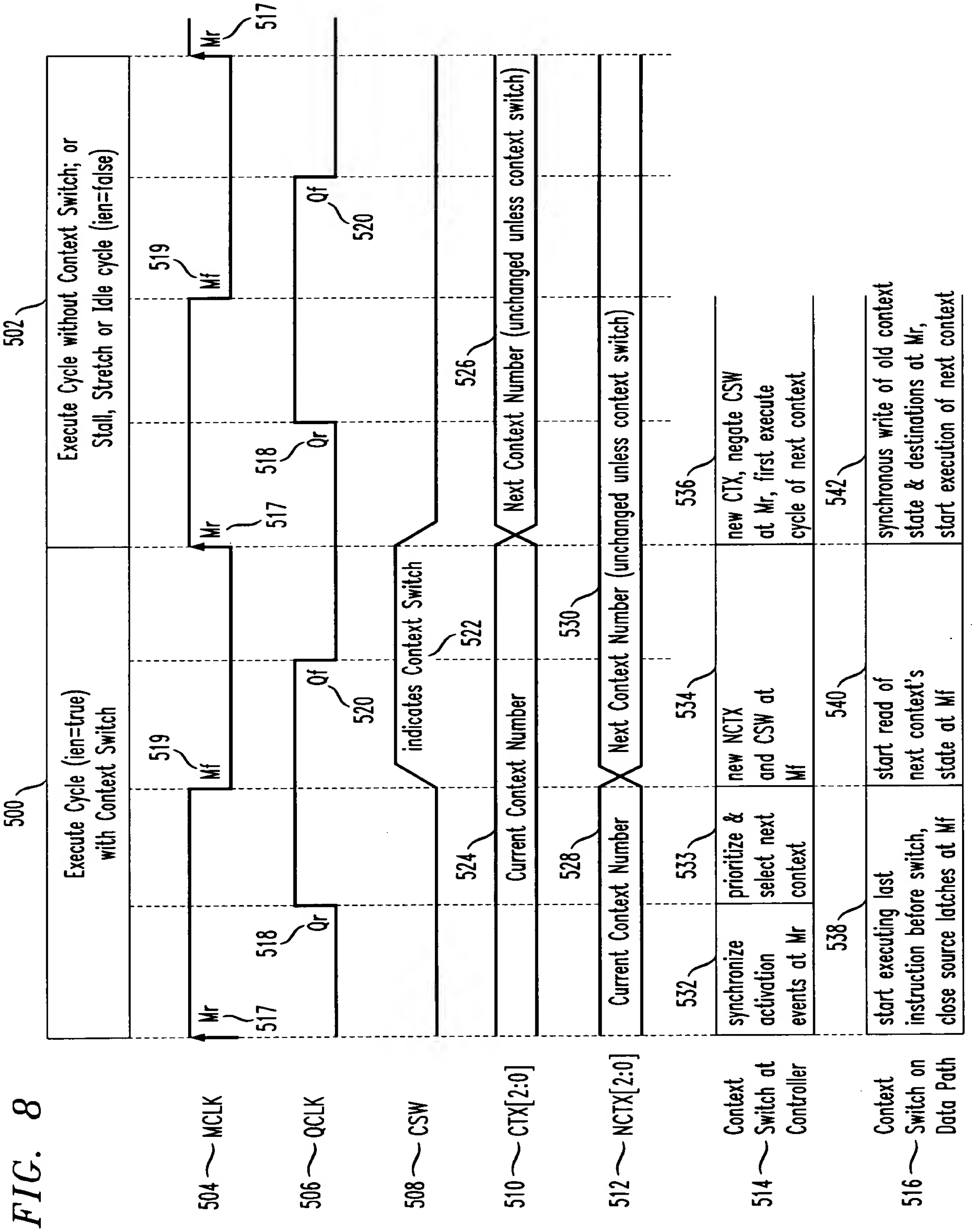

500
Execute Cycle (ien=true) with Context Switch
502
Execute Cycle without Context Switch; or Stall, Stretch or Idle cycle (ien=false)
504 MCLK
506 QCLK
508 CSW
510 CTX[2:0]
512 NCTX[2:0]
514 Context Switch at Controller
516 Context Switch on Data Path
Mr
517
518
Qr
519
Mf
520
Qf
522
indicates Context Switch
524
Current Context Number
526
Next Context Number (unchanged unless context switch)
528
Current Context Number
530
Next Context Number (unchanged unless context switch)
Next Context Number
532
synchronize activation events at Mr
533
prioritize & select next context
534
new NCTX and CSW at Mf
536
new CTX, negate CSW at Mr, first execute cycle of next context
538
start executing last instruction before switch, close source latches at Mf
540
start read of next context's state at Mf
542
synchronous write of old context state & destinations at Mr, start execution of next context

FIG. 9

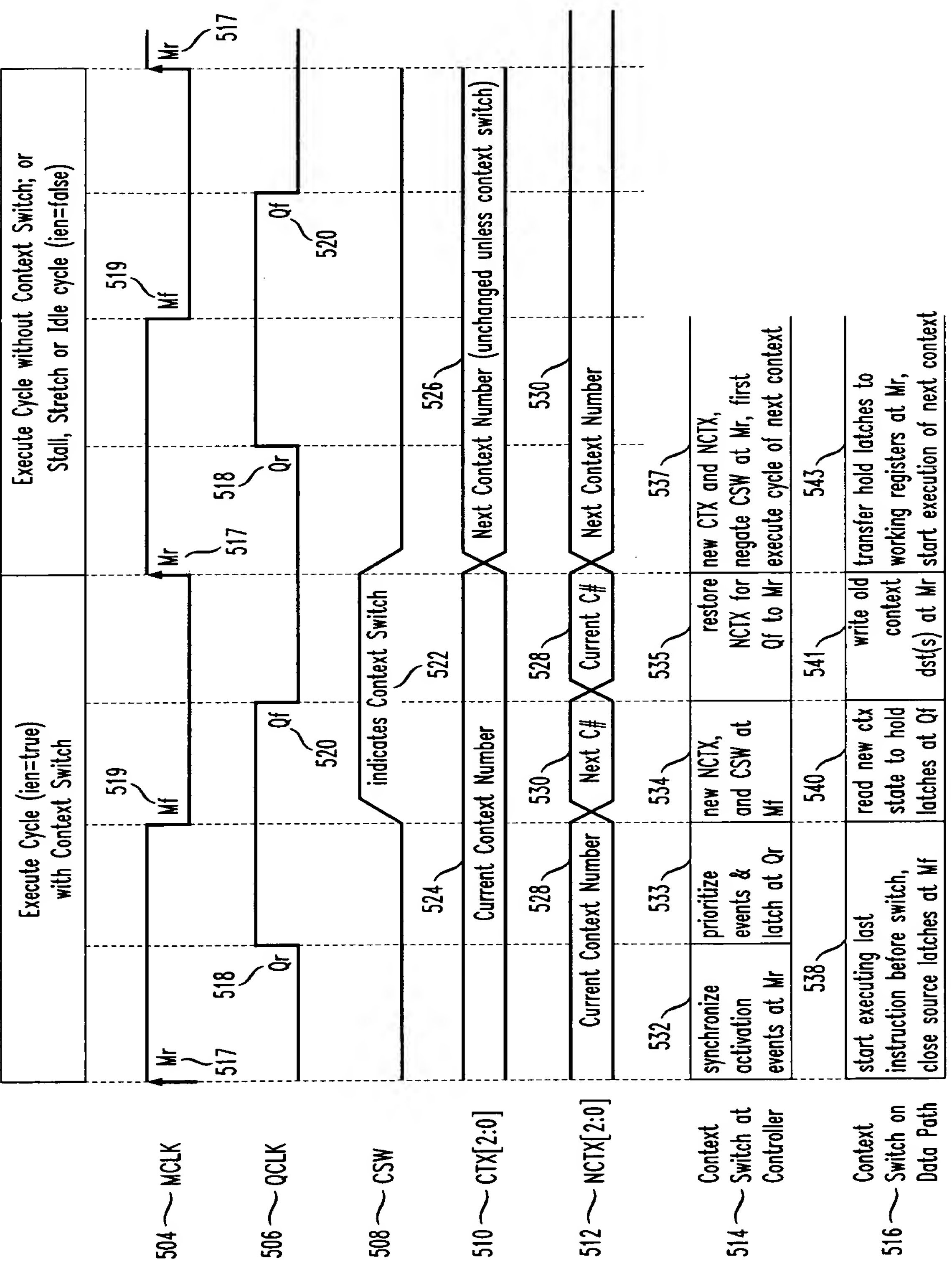
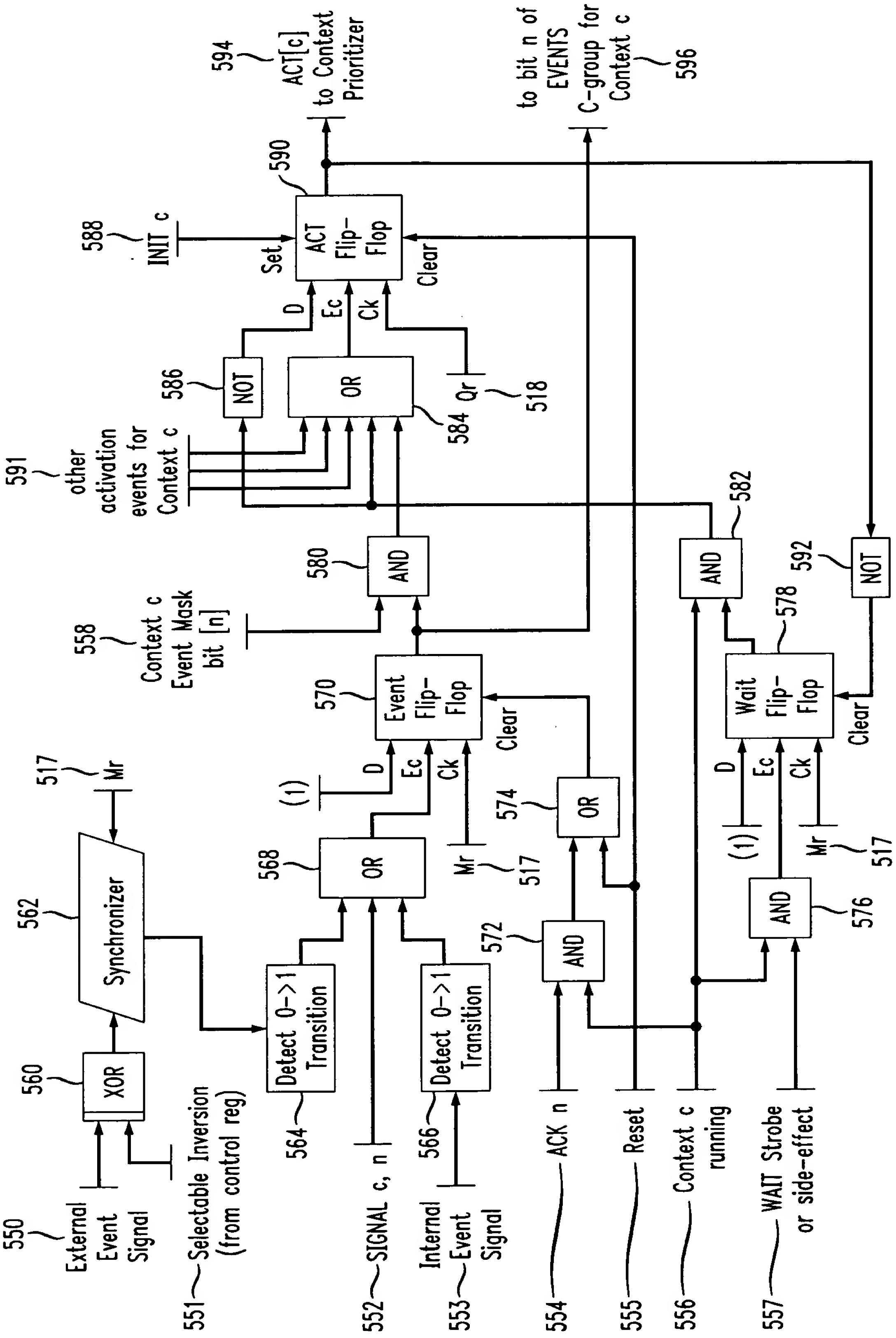

500
Execute Cycle (ien=true) with Context Switch
502
Execute Cycle without Context Switch; or Stall, Stretch or Idle cycle (ien=false)
504 MCLK
Mr
517
518
Qr
519
Mf
520
Qf
506 QCLK
508 CSW
indicates Context Switch
522
510 CTX[2:0]
524
Current Context Number
526
Next Context Number (unchanged unless context switch)
512 NCTX[2:0]
528
Current Context Number
530
Next C#
Current C#
Next Context Number
514 Context Switch at Controller
532
synchronize activation events at Mr
533
prioritize events & latch at Qr
534
new NCTX, and CSW at Mf
535
restore NCTX for Qf to Mr
537
new CTX and NCTX, negate CSW at Mr, first execute cycle of next context
516 Context Switch on Data Path
538
start executing last instruction before switch, close source latches at Mf
540
read new ctx state to hold latches at Qf
541
write old context dst(s) at Mr
543
transfer hold latches to working registers at Mr, start execution of next context

FIG. 10


594
ACT[c] to Context Prioritizer
to bit n of EVENTS C-group for Context c
596
590
588
INIT c
Set
ACT Flip-Flop
Clear
D
Ec
Ck
586
NOT
OR
584
Qr
518
591
other activation events for Context c
582
AND
580
AND
558
Context c Event Mask bit [n]
592
NOT
578
Wait Flip-Flop
Clear
D
Ec
Ck
(1)
Mr
517
570
Event Flip-Flop
Clear
D
Ec
Ck
(1)
574
OR
517
Mr
562
Synchronizer
568
OR
Mr
517
572
AND
576
AND
560
XOR
Detect 0->1 Transition
564
Detect 0->1 Transition
566
550
External Event Signal
551
Selectable Inversion (from control reg)
552
SIGNAL c, n
553
Internal Event Signal
554
ACK n
555
Reset
556
Context c running
557
WAIT Strobe or side-effect

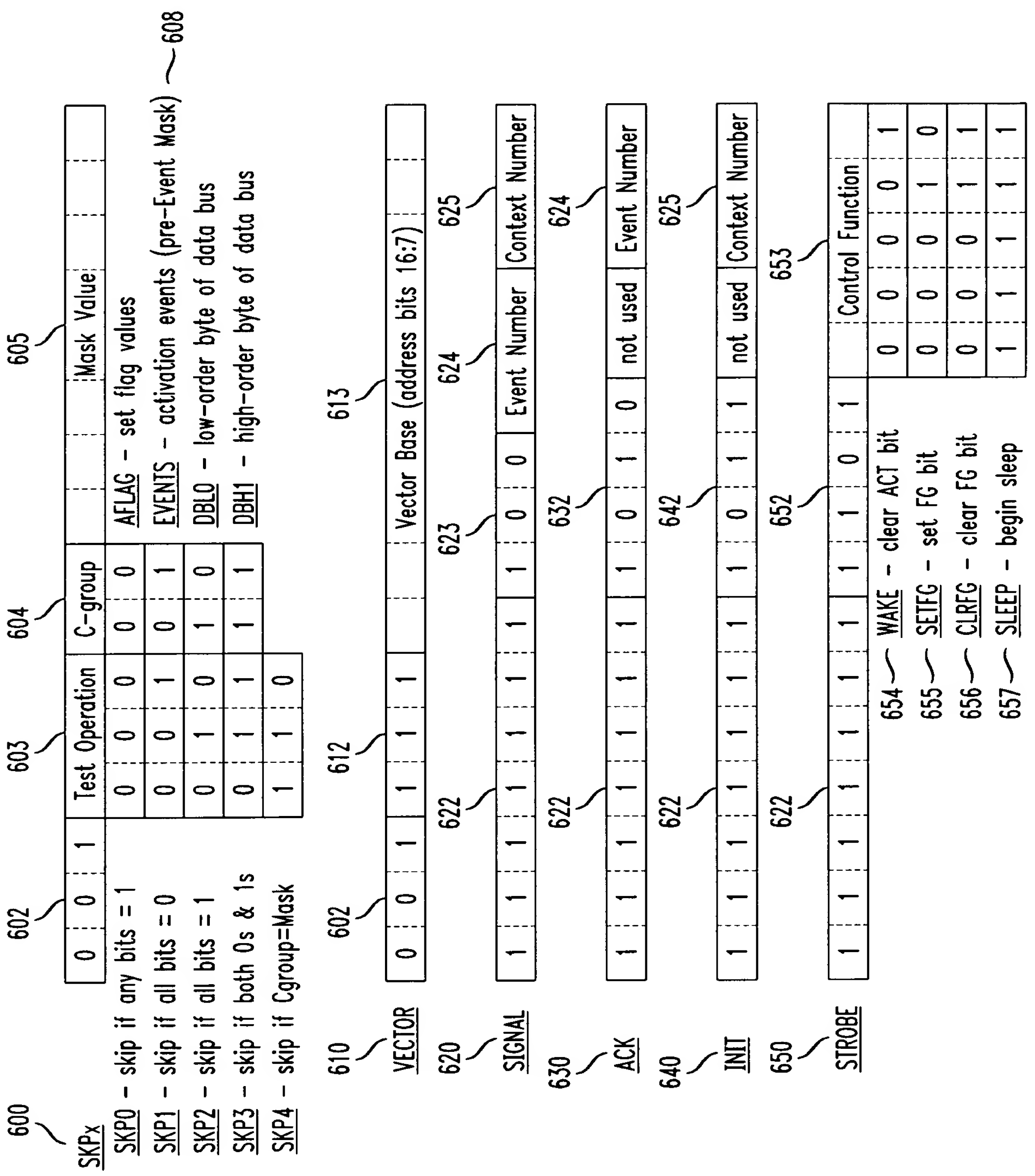

600
SKPx
602
0 0 1
603
Test Operation
604
C-group
605
Mask Value
SKP0 - skip if any bits = 1
0 0 0 0 0
SKP1 - skip if all bits = 0
0 0 1 0 1
SKP2 - skip if all bits = 1
0 1 0 1 0
SKP3 - skip if both 0s & 1s
0 1 1 1 1
SKP4 - skip if Cgroup=Mask
1 1 0
AFLAG - set flag values
EVENTS - activation events (pre-Event Mask)
608
DBL0 - low-order byte of data bus
DBH1 - high-order byte of data bus
610
VECTOR
0 0 1 1 1 1
612
613
Vector Base (address bits 16:7)
620
SIGNAL
1 1 1 1 1 1 1 1 0 0
622
623
624
625
Event Number
Context Number
630
ACK
1 1 1 1 1 1 1 1 0 1 0
632
not used
Event Number
640
INIT
1 1 1 1 1 1 1 1 0 1 1
642
not used
Context Number
650
STROBE
1 1 1 1 1 1 1 1 1 0 1
652
653
Control Function
0 0 0 0 1
0 0 0 1 0
0 0 0 1 1
1 1 1 1 1
654
WAKE - clear ACT bit
655
SETFG - set FG bit
656
CLRFG - clear FG bit
657
SLEEP - begin sleep

FIG. 11

FIG. 12

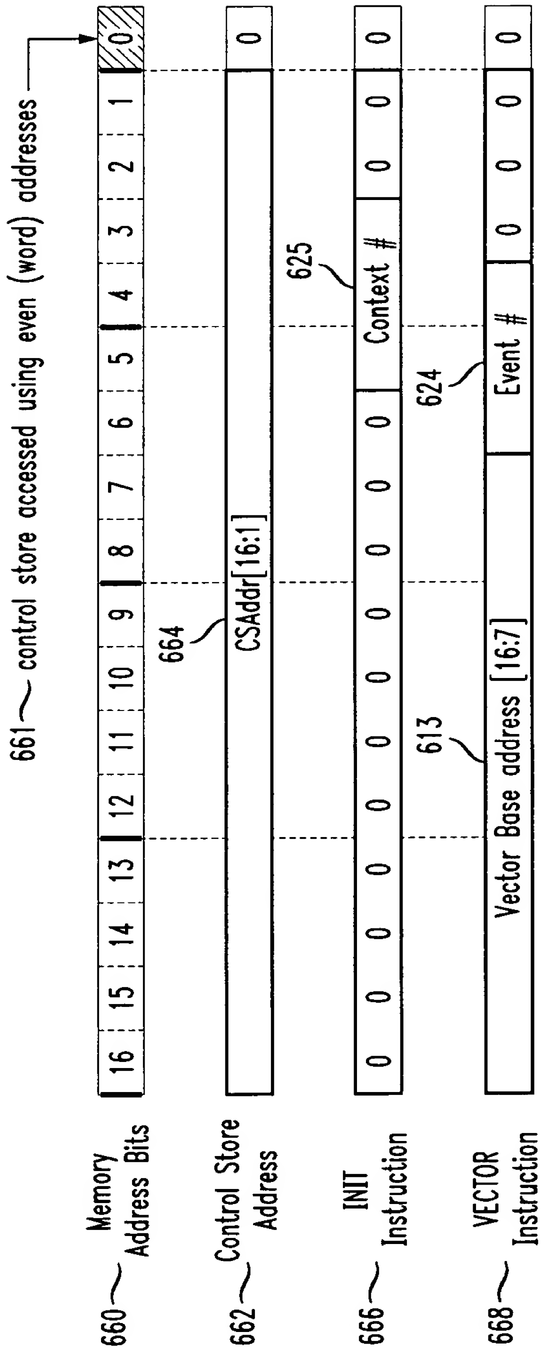

661 control store accessed using even (word) addresses
660 Memory Address Bits
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
662 Control Store Address
664 CSAddr[16:1]
0
666 INIT Instruction
0 0 0 0 0 0 0 0 0 0 0
625 Context #
0 0 0
668 VECTOR Instruction
613 Vector Base address [16:7]
624 Event #
0 0 0 0

FIG. 13

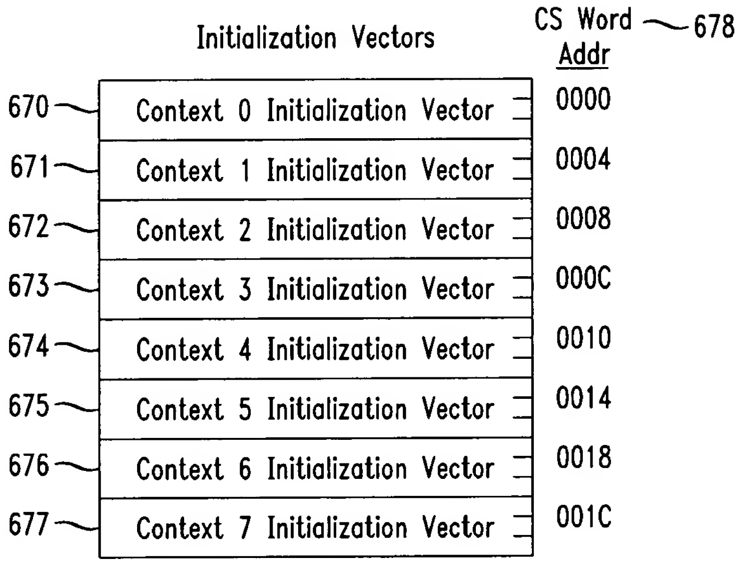
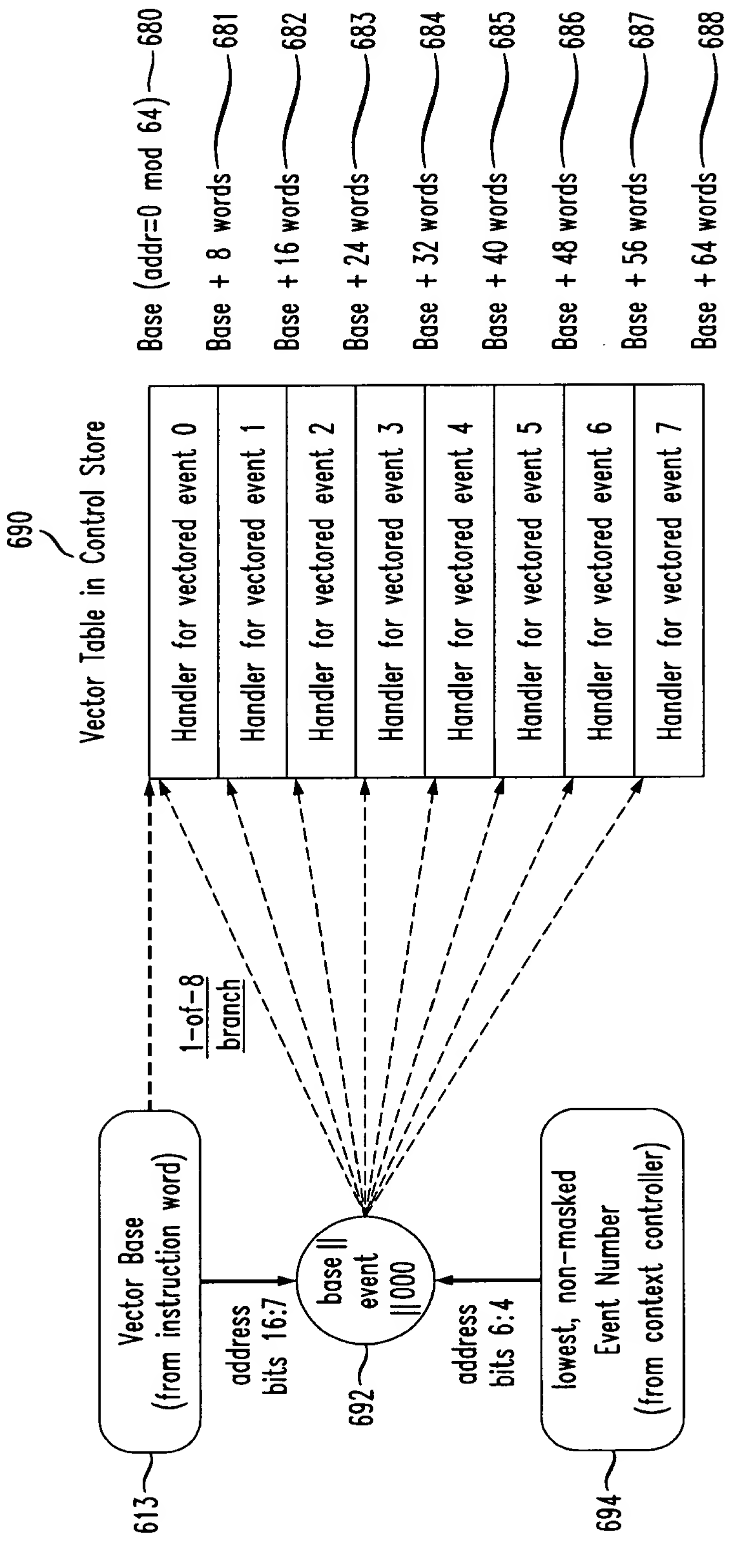

Initialization Vectors
CS Word Addr
678
670
Context 0 Initialization Vector
0000
671
Context 1 Initialization Vector
0004
672
Context 2 Initialization Vector
0008
673
Context 3 Initialization Vector
000C
674
Context 4 Initialization Vector
0010
675
Context 5 Initialization Vector
0014
676
Context 6 Initialization Vector
0018
677
Context 7 Initialization Vector
001C

FIG. 14


690
Vector Table in Control Store
Handler for vectored event 0
Handler for vectored event 1
Handler for vectored event 2
Handler for vectored event 3
Handler for vectored event 4
Handler for vectored event 5
Handler for vectored event 6
Handler for vectored event 7
Base (addr=0 mod 64) 680
Base + 8 words 681
Base + 16 words 682
Base + 24 words 683
Base + 32 words 684
Base + 40 words 685
Base + 48 words 686
Base + 56 words 687
Base + 64 words 688
1-of-8 branch
Vector Base (from instruction word)
613
address bits 16:7
base \|\| event \|\|000
692
address bits 6:4
lowest, non-masked Event Number (from context controller)
694